Resource Management for Enhancing Predictability in Systems with Limited Processing Capabilities

Alejandro Alonso
Dept. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
aalonso@dit.upm.es

Emilio Salazar
Dept. Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
esalazar@dit.upm.es

Jorge López
Dept. Arquitectura y Tecnología de Sistemas Informáticos
Universidad Politécnica de Madrid
jlopez@datsi.fi.upm.es

Abstract

There is an increasing demand for computing systems composed by heterogeneous computers, connected by different types of networks, and that allow for accessing a wide range of services in a seamless way. Some of those computers are mobile or embedded and have limited resources, and can be overloaded when trying to handle their users demands. Then it is not possible to ensure a proper behaviour of the running applications. This can be an important problem when dealing with critical events in healthcare, home surveillance, or forest monitoring.

Resource reservation is a valid basis for handling this issue. It allows for guaranteeing a certain resource share for applications that are important for the proper behavior of a given system. This paper describes an implementation of a resource management component and its integration in the Linux kernel. This piece of software has allowed to assign CPU budgets to standard Java threads, which is an important facility, given the widespread of this programming language. This implementation has been validated on service oriented middleware, where relevant services are executed by thread with guaranteed budget, to improve its predictability.

1 Introduction

Computer systems are increasingly part of our daily life. They are integrated into everyday objects and activities. It is common to carry one or more devices with computing and communication capabilities, or to rely on computers for the correct behavior of our daily used appliances and objects. This also implies that there are important requirements on their correct behavior.

The current trend is to use these devices as the mean for accessing a wide range of remote services, interact with powerful servers, or get contextual information from other devices. There are a large number of efforts trying to facilitate the development of such type of applications. Some examples can be found by just looking at research projects funded by the European Union, such as:

- MORE [12]: This project has developed a network-centric middleware intended to develop heterogeneous distributed systems that must collaborate in order to provide a given service to the end user. Its main features are to facilitate communication and distributed intelligence between groups and users, to provide a middleware appropriate for embedded systems that facilitates the relation with persons, and to provide an API that hides the underlying complexity in the communication between embedded devices to the developer.

- SMEPP [17]: This project has developed a network-centric middleware oriented towards peer-to-peer embedded systems. In this scenario, all the elements of the network are symmetrical. The middleware should be secure, generic and highly customizable, allowing for its adaptation to different devices (from PDAs and new generation mobile phones to embedded sensor actuator systems) and domains (from critical systems to consumer entertainment or communication).

- EMMA [3]: This project tried to deal with the trend that networks link smart devices, in addition to people. These devices share information, knowledge and services anywhere anytime. These networks of smart devices can be mission critical for application domains, such as transport. The project will developed a middleware intended to hide the complexity of the underlying infrastructure while providing open interfaces to third parties. It also fosters cost-efficient ambient intelligence systems with optimal performance,

high confidence, reduced time to market and faster deployment.

- HYDRA [6]: This project aimed at providing a middleware that allows the incorporation of heterogeneous physical devices into their applications by offering easy-to-use interfaces for controlling any type of physical device. The middleware intends to allow transparent communication means and to support different types of communication protocols. It is based on web services and provides means for basic operations, such as device and service discovery, P2P communications and diagnostics.

The type of applications targeted by these projects has some common characteristics, such as : i) enabling the connection to a potentially large number of embedded, mobile or powerful computers, ii) getting information from different devices, for composing a relevant view of a given context, iii) detecting and handling meaningful events, iv) showing the information and services that the user is interested anytime, v) allowing access to all this facilities from mobile devices.

These mobile and embedded devices use to have limited resources, which implies that there are situations where it is not possible to ensure a proper behavior of all the running applications. This is the case when the resources required are larger than those available. In this situations, there are critical applications that may behave in an undesirable way.

There are initiatives for using this type of systems for supporting applications with safety and dependability requirements, such as healthcare, forest monitoring, or home control. In these applications, there are events that must be handled in a predictable and dependable way. As an example of a healthcare application, the mobile phone can be connected with devices for measuring blood pressure or blood glucose level for patients with diabetes. These measurements can be sent to a hospital, in order to follow the patient evolution remotely. If a certain value means a potentially dangerous situation for the patient, it should be handled properly. There are other examples, such as fires detected by forest monitoring equipment, alarm situations on home surveillance systems, etc. In summary, there is a trend to develop applications that will help to improve home safety, prevent or attenuate natural disasters, and extend healthcare assistance. In some cases, they have to react in a predictable way in order to deal with potentially dangerous situations.

The traditional approach for providing a safe, predictable, and timely behavior is to use specific operating systems, communication protocols, and programming languages, with real-time and safety features. However, this alternative poses additional problems, as they may prevent the use of development tools of common use (libraries, protocol stacks, languages, etc.), that in some cases do not have a real-time/safety counterpart.

The purpose of this paper is to describe the experience of providing resource management facilities on a standard Java Virtual Machine. In particular, a CPU budget is assigned to relevant threads. In this case, resource management is implemented on top of standard operating systems and programming languages, while improving the reliability and predictability of the applications. The basis is to guarantee some resource usage percentage, that would applications to perform their main duties in a predictable way.

This paper describes a simple framework that has been developed for supporting resource budgets. The main emphasis in its design an implementation has been simplicity, efficiency, and facilities for experimenting with different resource management approaches. This framework has been put together with a standard Java Virtual Machine (JVM), in order to provide CPU reserves to standard Java. Finally, an use case is used to validate the interest of the approach, that has been developed in the context of the MORE project.

1.1 Related Work

The concept of using resource shares for improving predictability was first introduced in the RT-Mach kernel [10] [11]. The accounting facilities were tightly coupled with the kernel. Portable RK [14] provided an implementation that was more independent from the kernel, to facilitate its porting to other platforms.

The interest of the topic has motivated a large number of works. As an illustration, The FRESCOR project [4] supports adaptive real-time systems by by creating a contract model that specifies which are the application requirements with respect to the flexible use of the processing resources in the system. There is quality of service management that adapts itself to the application-domain concepts of quality, requesting the required information from the underlying system in a transparent way. The OCERA project [13] provides an integrated execution environment for embedded real-time applications. It is based on components and combines the use of two kernels, Linux and RTLinux-GPL to provide support for critical tasks (RTLinux-GPL executive) and soft real-time applications (Linux kernel). These works provide advanced means for dealing with quality of service and resource management, although they usually provides a more complex framework than required for the type of systems targeted in this work. In addition, they are usually done for a particular operating system.

There are a number of additional works, such as [15], [9], or [2]. However, they are target towards a particular functionality and more complex than required in this framework. The aim of the work presented in this paper was to make the simplest possible component for resource usage accounting and enforcing. Portability and reusability are topics of primary concern. The goal is to port or integrate the proposed framework in existing operating systems and to experiment with higher level and more complex budgets models, which does not need to

be included in the kernel and that does not depend on a particular operating system, as they only interact with the mentioned resource manager.

2 Developing Applications with Dependability Requirements

Safety systems [8] are usually defined as those which failure may cause important damages to the environment and to persons. However, the development of safety systems implies a set of very strict design and development rules. Among them, it can be required the use of specific language subsets that allows for performing a set of exhaustive tests to ensure proper system operations. There are language features that prevent some of these tests and must be avoided. There are safe subsets of programming languages such as Ada. With respect to Java, there is currently a working group defining a safe language profile. However, this type of subsets limits the range of applications that can be developed. Then, they are not suitable for systems where applications with some safety requirements coexist with standard ones.

Time requirements are also common in systems with some safety implications. Real-time systems are those which correctness depends not only on their outputs, but on the time when they are produced as well. A good result that is obtained late can be a failure in this type of systems. It is important to note that the main issue is not about performance, but on predictability of the timeliness of the results. The applications sketched before may also have time requirements.

The development of real-time systems highly predictable implies the use of specific techniques and protocols: languages with appropriate programming constructs, operating systems with real-time scheduling algorithms, communication protocols, careful designs avoiding unbounded delays, etc. However, the use of this technology would have implications when developing the type of systems targeted at this paper. For example, several libraries of common use are not available for them, real-time communication protocols are only used in specific domains, and there is lack of proper standards. On the other hand, in the type of applications subject of this work, time requirements use to be soft: a occasionally missed deadline causes little harm.

The use of modern resource management techniques can be used for improving system predictability. One helpful facility is the use of resource budgets, which can be referred to CPU, memory, bandwidth, etc. In this paper, emphasis is put on CPU budgets, that are defined by three main parameters:

- Usage time: Maximum time that a thread (or cluster of threads) is allowed to use the resource associated to the budget, in this case the CPU, with the given priority.
- Period: Usage time is replenished at a fixed rate, defined by this parameter.
- Priority: Relative urgency of a real-time thread.

If the budget is exhausted, some actions must be taken in order to ensure that other threads can use its budget. Real-Time Java Specification provides real-time threads with a behavior similar to that described. An important innovation of this work is using this facility on a standard Java Virtual Machine (JVM).

The approach relies on an implementation of CPU budgets, that is described in the following section. It wprovides a C interface and relies on some real-time facilities available in the Linux kernel. In the use case, it is used JamVM, which is an open source Java virtual machine and, after a careful study of the source code, it was noted that there is a one-to-one relation between a Java thread and a Linux (POSIX) thread. Hence, the approach would be to assign the budget to Java thread. Some successful experiments have also been done with the Sun JVM. It appears that also follows the one-to-one model, although it has not been possible to check with absolute confidence this fact.

3 Resource Management

The motivation of the work described in this paper was to develop support for resource budgets, in a way that is simple and portable. Simplicity was intended for making it efficient and for being able to develop on top of it budget models with higher abstraction level. This has converted this platform in an excellent mean for experimenting with different approaches for resource usage models, without the need for developing from scratch support for them. In addition, it has been possible to use it in different scenarios, due to its facilities for being ported and adapted.

The resource management consists of two main components:

- BACC (Budget Accountant): Provides the basic means for enforcing and accounting for resource usage. It does not include the policies for budget replenishment or handling overruns. In this way, it is feasible to experiment with different options.
- RTC (Run-Time Control): Provides a higher-level API than the BACC. It provides periodic budget replenishment and a particular way of handling overruns, based on reducing the thread priority until budget is available again.

This section describes the main design goals and decisions taken with respect to these modules.

3.1 Budget Accountant

The main functions provided by the Budget Accountant (BACC) are the following:

- Management of task budgets: The BACC allows for the creation/deletion of budgets. They are associated to a particular thread or to a cluster of threads. As budget replenishment is not automatically provided, a function is provided for calling this operation from an external component. In this way, it is possible to experiment with different models. In addition to the mentioned operations, budget parameters can be modified dynamically. Changes are considered at the beginning of the next refill period.

- Overrun detection. The overrun handling mechanisms should be as simple and flexible as possible. The BACC does not decide on the corrective actions to take. The BACC interface allows for two possible options for dealing with overruns. One is to skip them. The other is to perform a call to a function provided by the programmer. In this way, this event is notified to a higher level software component, which will take appropriate actions to handle it.

- Statistical information. The BACC keeps statistical information in order to let monitoring applications to check how a particular thread is behaving: whether the assigned budget is appropriate for the functionality to provide, how frequent are overruns, or which is the mean percentage of the budget used. There is an API that allows for activating and deactivating budget accounting for statistical information, to retrieve it and to reset this data.

The BACC accounting requires to get an external event whenever a meaningful change in the task that is using a resource occurs. Let suppose that the BACC is charging the use of a resource to a specific thread. When the task using this resource changes, the BACC needs to know it in order to update the remaining resource usage time of the previous task, and taking the required actions to start charging the use of the resource to the new one and to detect possible overruns. The most appropriate way to handle this event is to arm a timer with the remaining usage time of a task, when it starts using a resource. If the timer expires before the task is deactivated, then an overrun has occurred. In the case of the CPU, the event is a change of context. The specific low level routines for resource accounting are in charge of detecting the appropriate event for each resource and notifying it to the BACC.

As a summary figure 1 shows the main events and interactions in the BACC operation.

3.2 Run-Time Control

The Run-Time Control (RTC) is a software layer that extends the basic functions provided by the BACC. In particular, it makes available a complete, whilst simple, support to CPU budgets. It relies on the BACC and adds two main features:

- Periodic budget refilling. The BACC does not perform by itself budget refilling. It provides the basic mechanism, by including a function in its API, that must be called externally. The RTC refills the active budgets in a periodic fashion. When creating a budget, it is needed to indicate the associated thread(s), the resource usage time, the priority and the refilling period. The RTC invokes the mentioned function periodically, to ensure that the defined utilization time is assigned to the thread.

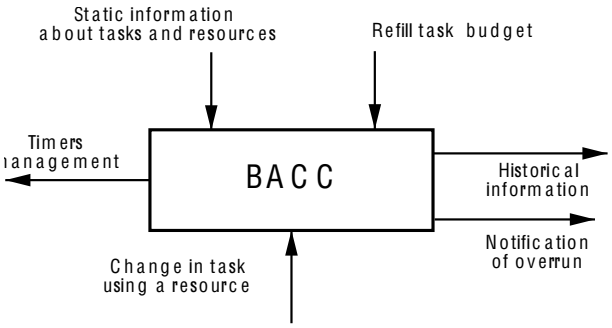

Figure 1. BACC Events and interactions

- Overrun handling: This is an important functionality, as it is mandatory for guaranteeing CPU budgets. When an overrun is detected, the main operation to perform is to prevent the corresponding thread to continue using the CPU, at least with its original priority. Once again, the policy provided by the RTC is simple and efficient. When an overrun exists, the priority of the thread is lowered to the minimum available one. In this way, it is ensured that it will not preclude other threads from using their budgets, although it is possible to use spare computation time when available.

  This behavior is appropriate for systems targeted in this work. Other alternatives, such as killing the thread causing the overrun, are not required. However, it is always possible to restrict the execution of threads trying to use the CPU more than budgeted in a persisting way, by analyzing the statistical information provided by the BACC.

4 Resource Management in Linux

This section describes the porting of the BACC and RTC to the Linux kernel. If the system reserves resources to some applications it is because they are considered more important and their execution improves the final user satisfaction. Then, a reasonable behavior for threads with CPU reserves is to let them execute before or with higher priority than the others.

The basic Linux scheduler is based on the well-known quantum technique [1]. CPU time is divided into time slices, and every process has a specified one. If the process is not terminated or self-suspended when its current slice expires, a process switch occurs and the CPU is granted to another ready process. Processes are ranked according to

priority, which is a process parameter which may change dynamically according to complex algorithms. This type of scheduler is clearly not appropriate for the intended behavior.

Fortunately, Linux provides another type of scheduler, in compliance with the POSIX standard [7]. The alternate scheduler is identified as `SCHED_FIFO`, and it is priority-based and preemptive. For each process, it is possible to specify the scheduling policy to be used and a fixed real-time priority. Real-time priorities are always more urgent than non-real-time priorities. The threads with available budget receives real-time priorities, so they have preference for using the CPU.

Given the required operation for the RTC and BACC, they should be included in the kernel. The BACC has to execute whenever there is a context switch or the accounting timer expires. The Linux scheduler was slightly modified to call the relevant function of the BACC when that event happens. It is obvious that keeping the BACC out of the kernel will increase the overhead.

In addition to provide a higher level interface to the accounting facilities, the RTC performs the periodic refill of budgets and the handling of the budget overrun as described previously. The first operation is executed when the accounting timer expires. The overrun handling is executed as a request of the BACC. For these reasons, it was also advisable to include the RTC in the kernel. In this way the implementation was easier and the overheads were reduced, as a number of switches from the kernel mode to user mode will be required otherwise. The final design structure is depicted in figure 2.

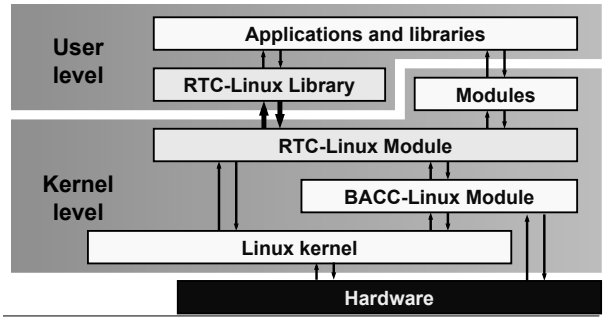

Figure 2. Components in the BACC implementation in Linux

The resolution of the Linux timer recent versions of the kernel, it is enough for the requirements in this work. However, it requires special hardware that older PCs does not include. For this reason, it was kept an updated implementation of precise timers within the kernel, that relies on the timer counter in the Pentium IV and later. Then, it is possible to use older hardware, which is convenient for some of the intended uses of this piece of software.

The communication between the user and kernel parts of the RTC is done by means of a dynamic file system. In particular, it is used the `procfs`, which is usually located in `/proc`. In order to access to the RTC, it has been created the directory `/proc/driver/rtc-linux`, where resides the special file for issuing commands to the RTC, plus some additional files with information of the state of the modules, which are useful for debugging purposes. The specific commands are called using the standard `ioctl` operation.

CPU budgets are assigned to Java threads in a straightforward manner. If a Java thread has a CPU budget, it gets the corresponding PID of the Linux thread by using native methods to invoke kernel operations. After that, it invokes the proper operations in the RTC API, following the mentioned approach.

5 Test and Validation

5.1 Testing of CPU Budgets

The efficiency and proper behavior of the implementation has been checked with a set of exhaustive tests. The proper behavior of the Linux modules (BACC and RTC) has been carried out by a set of tests in different programming languages: Ada, Java y C.

Figure 3 shows the execution of a synthetic load composed by a real-time thread (RTC, continuous line) and another standard POSIX thread (NO_RTC, dotted line). The code executed in both cases is the following:

```
for i in 1 .. 50000000 loop
    value := Sqrt (largeNumber);
    value := value * value / (value - 1.0);
end loop;
```

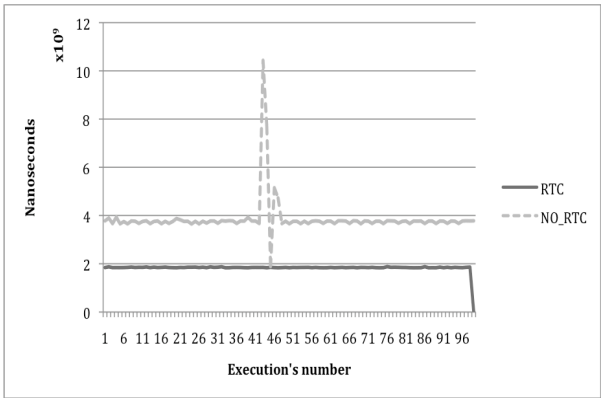

Figure 3. Comparison of the execution of standard and BACC threads

It can be observed that the execution time of the real-time thread is nearly constant, very stable and more efficient. It is possible to see the interference of other activities of the system over the standard Linux thread. In the case of the BACC thread, the execution time is more or less constant over the different samples taken. This test

has been repeated a high number of times, with similar results.

This test shows that the CPU budgets works properly in a system with with few load. However, the main goal of the BACC is to ensure the execution of threads with time requirements in a more predictable way, even in situations when the system is overloaded. In order to assess such behavior, a testbed has been created where the thread under observation runs the code shown above, in concurrency with five hundred threads running the same code, along with additional system activities such as the compilation of a large system. The overall system load was close to 100%. Figure 4 shows that the threads with CPU budget run properly.

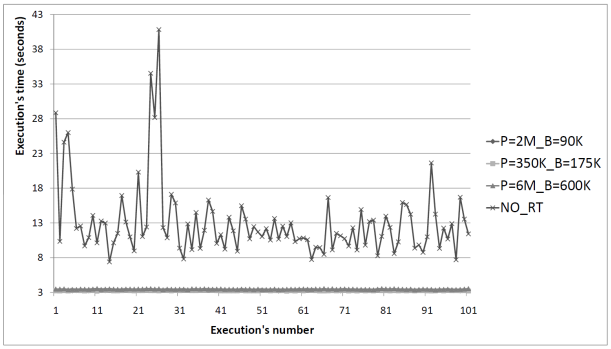

Figure 4. Execution of standard and BACC threads

Results are much better for the BACC threads than for standard thread. It can be observed in the figure that the execution time of the BACC thread is nearly constant. The variation with respect to the previous case is less than 1%.

On the other hand, standard Linux threads suffer a high interference from other threads. Their mean execution times are raised up to 300% with respect to the values measured with low system overhead. In addition, response times are not very predictable, as the different between the maximum and the minimum is about three orders of magnitude higher than in the case of the BACC thread.

In summary, the execution of the BACC is correct and BACC threads shows a much better behavior in terms of time response and predictability.

5.2 Use Case Evaluation

This use case has been performed in the context of the MORE project. The Core Management Service (CMS) is the most important component in the MORE middleware. It relies on the Service Oriented Architecture. The CMS is the central component. All nodes must execute this service. From the behavioral point of view, it relies on a thread pool for the execution of the remote invocations to services in the given node. An in-depth study of the code, shows that it is based on a set of threads that are created at system initialization, which aim is to execute clients' requests. These threads are suspended, until a new request arrives. Then, one of them is activated for handling it. Finally, the requested service is executed. Given that all of this handling is performed by the same thread, the class that serves for creating them has been modified in order to use the threads provided by the BACC, instead of the standard ones.

The ThreadPool class includes the implementation of a threads pool that uses the CMS for handling the SOAP messages that are received and the responses. It is a class that belongs to Apache org.mortbay.jetty package. It is called from the implementation of the DPWS [16] [5]in the core.

A class has been included that encapsulates the Java code required for performing calls to the methods in the RTC module, which is the external interface to the CPU budgets, and that is coded in C. This code, that has a very low level, performs type conversions and parameter passing, required for the use of the shared libraries.

In summary, the most important modification has been the change of the ancestors of the ThreadPool class, in order to extend the BACC threads in Java, instead of the native ones. BACC threads requires a more complex initialization. Special code has been added for hiding this features to the applications developer.

The software configuration of the tests consists of a computer that provides a set of services, some of them with associated time or safety requirements. These services execute a synthetic load, similar to that shown in the previous section. A client is running in another computer. It performs periodic invocations to the mentioned service. The overall system load is close to 100%.

These tests have been executed in different computers. The figures shown in the paper were obtained on a execution platform characterized by a Pentium IV at 3GHz and 1GB of RAM memory. The operating system was a Linux Debian with a 2.6.21 version of the kernel. It was not possible to use JamVM in these experiments because of problems encountered when using some libraries required by the MORE middleware. Instead, it was used a Sun JVM 1.6.0.12.

A summary of a large set of executions can be shown in figure 5. It can be observed that the execution time of the services associated with threads with CPU budget is not as clean as in the previous experiment. There are peaks that implies larger response time than desired. This is most probably due to the garbage collector and some other internal operations of the JVM. The most obvious way to deal with this issue is to use a RTSJ compliant virtual machine, with a proper real-time operating system underneath. However, as it has been previously mentioned, this option is not viable for a large number of systems, as it precludes the use of class libraries of common use and requires of developers specialized in this development platform.

However, the results are much better than when CPU

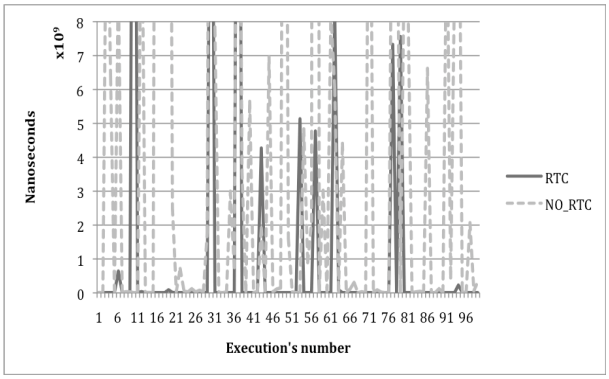

Figure 5. Original and modified CMS (RTC)

budgets are not used. If the mean response time is considered, when standard Java threads are used, the mean execution time is 15% than the worst cases in an 80% of the executions. On the other hand, when threads with CPU budgets are used, this figure is reduced to only a 20% of the total experiment executions.

In summary, it is possible to conclude that the use of CPU budgets improves the predictability of the service provision, reducing by a factor of four the number executions that are not acceptable due to its large response time. However, it is necessary to continue finding ways of improving this figures using an standard JVM. The use of programming models that reduce the intervention of the garbage collector, will be of interest. The use of open JVM will help to better understand the internal behavior of the system and to try to modify it to avoid undesirable interferences.

6 Conclusions

This paper describes the current state of the design and implementation of two simple and portable components for implementing resource budgets, which are a very interesting way of enhancing predictability on systems running on standard operating systems. The proposed framework allows the definition of resource budgets, associate them to real threads, and to account for resource usage and guarantee budgets at run-time.

This approach has been tested on the MORE middleware, in such a way that threads running critical services. CPU budgets are used as a mechanism for improving the response time predictability in the system. BACC and RTC are two Linux modules that allows to guarantee CPU shares to threads. A number of classes has been developed in order to access these facilities from Java threads, for providing some real-time support to urgent services, within the limitation of the used execution platform.

References

[1] B. Daniel P. Bovet, M. Cesati, Understanding the Linux Kernel, 3rd Ed., ISBN: 978-0-596-00565-8, O'Reilly, 2006.

[2] S. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes. Proceedings. 24th IEEE International Real-Time Systems Symposium, pages 396407, December 2003.

[3] EMMA web site, http://www.emmaproject.eu/

[4] FRESCOR project web page, http://www.frescor.org

[5] Hyung-Jun Yim, Il-Jin Oh, Yun-Young Hwang, Kyu-Chul Lee, Kangchan Lee, Seungyun Lee Design of DPWS Adaptor for Interoperability between Web Services and DPWS in Web Services on Universal Networks, International Conference on Convergence Information Technology, 2007.

[6] HYDRA web site, http://www.hydramiddleware.eu

[7] 9945-2009 ISO/IEC/IEEE Information Technology Portable Operating System Interface (POSIX) Base Specifications, Issue 7

[8] I. Lee, J. Leung, S. Son. Handbook of Real-Time and Embedded Systems. Chapman Hall, 2008

[9] A. Marchand and M. Silly-Chetto. Qos and aperiodic tasks scheduling for real-time linux applications. 6th Real Time Linux Workshop, November 2004.

[10] C. W. Mercer, S. Savage, and H. Tokuda, Processor capacity reserves: an abstrac- tion of managing processor usage, in Proceedings of 4th Workshop on Workstation Operating Systems (WWOS-IV), 1993.

[11] C. W. Mercer, R. Rajkumar, and J. Zelenka, Temporal protection in real-time operating systems, in Proc. of 11th IEEE Workshop on Real-Time Operating Systems and Software, 1994, pp. 79-83.

[12] MORE project web, http://wwws.ist-more.org/

[13] OCERA project web page, http://www.ocera.org/

[14] S. Oikawa and R. Rajkumar, *Portable RK: A portable resource kernel for guaranteed and enforced timing behaviour* in IEEE REAL-TIME TECHNOLOGY AND APPLICATIONS SYMPOSIUM, pp. 111-119, IEEE Computer Society Press, 1999.

[15] N. Perng, C. Liu, and T. Kuo, "Real-Time Linux with Budget-Based Resource Reservation", J. of Informacion Science and Engineer., 22, 31-47, 2006.

[16] A. Sleman, R. Moeller, "Integration of Wireless Sensor Network Services into other Home and Industrial networks; using DPWS", International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA 2008.

[17] SMEPP web site, http://www.smepp.org